METHOD AND SYSTEM FOR SEMANTICALLY LABELING STRINGS AND PROVIDING ACTIONS BASED ON SEMANTICALLY LABELED STRINGS

Technical Field

This invention relates to document creation program modules. More particularly, this invention relates to a method and system for semantically labeling strings of text during creation of an electronic document and providing a selection of actions that may be performed based on the semantically labeled strings.

Background of the Invention

Electronic documents such as word processing documents and spreadsheet documents typically include semantic information that would be helpful if the information was recognized as such. Recognition and use of this semantic information could result in increased interoperability between desktop software applications and other desktop applications and/or web-based applications.

The ability to recognize strings of text, such as in search engines, is well-known. Additionally, various information retrieval systems have capabilities to label documents. For example, the LEXIS-NEXIS service provides links in some of its documents for company names, famous people and stock ticker symbols.

However, none of the services described above allow strings of text to be labeled with semantic information on-the-fly,

i.e., as a user is typing text into a document and creating a document. Thus, there is a need for a method and system for semantically labeling strings while a user is creating a document and providing user actions based on the type of semantic label applied to the string.

In addition to the need described above, there are other needs associated with electronic document program modules such as word processing program modules and spreadsheet program modules. There is the need for tighter integration between electronic document program modules and personal information manager program modules. For example, a user often wants to insert an address stored in a contacts folder of a personal information manager into a word processing document, such as when writing a letter. This insertion may be time-consuming. Typically, the insertion requires the user to switch from a word processing program module to the personal information manager program module, open the contacts folder, find the appropriate person in the contacts folder, copy the address from the contacts folder, switch back to the word processing program module and paste the address into the document at the appropriate place. Thus, there is a need for a tighter integration between information stored in personal information manager program modules and information needed by electronic document program modules.

There is also the need for tighter integration between web properties and the content of electronic documents. For example, sales people sometimes need directions to a client's site. The address of the client's site may be included in an e-mail message or electronic document. However, getting directions to the client's site via an Internet mapping service requires navigating to a website, entering the elements of the address into the appropriate search fields (e.g. street address, city, state, zip) and executing a search for the correct map. Thus, there is the need for tighter integration between web properties and the content of electronic documents.

There is also the need for integration between information on an Intranet and the content of electronic documents because different organizations have different types of data they deem important. Certain industries, services and professions may have words which are commonly found in documents. For instance, in drug companies, chemical names are extremely important and there's likely to be a myriad of information about a particular chemical name on a drug company's Intranet. For car companies, the important data might be model names or part numbers. For universities, the important data might be names of professors, students or courses. The prior art services do not allow customization of strings that may be recognized. None of the prior art services allow third parties to provide a recognition program module so that these customized words may be recognized and an action program module so that appropriate actions may be taken based upon these recognized words. Thus, there is the need for integration between important data on an Intranet and the content of electronic documents.

Thus, there is a need for a method and system for semantically labeling strings while a user is creating a document and providing user actions based on the type of semantic label. There is a further need for a tighter integration between information stored in personal information manager program modules and information needed by other program modules. There is also the need for tighter integration between web properties and the content of electronic documents. There is the further need for integration between important data on an Intranet and the content of electronic documents.

Summary of the Invention

The present invention satisfies the above described needs by providing a method and system for semantically labeling strings of text during creation of an electronic document and providing a selection of actions that may be performed based on the semantically labeled strings.

In one aspect, the present invention provides a computer-implemented method for semantically labeling strings during creation of an electronic document. When an application program module receives a new string, such as when the user enters a new paragraph or cell value into an electronic document or edits a previously entered paragraph, the paragraph containing the new string is passed from the application program module to a recognizer DLL. The recognizer DLL is responsible for communicating with the application program module, managing the jobs that need to be performed by the recognizer plug-ins, receiving results from the recognizer plug-ins and sending semantic category information to the application program module.

During idle time, the paragraph is passed to the recognizer plug-ins. The recognizer plug-ins are executed on the paragraph to recognize keywords or perform other actions defined by the recognizer plug-in. As part of executing the recognizer plug-in, the paragraph or cell value may be broken into sentences by the recognizer plug-in. However, each recognizer plug-in is responsible for its own sentence-breaking. After the keywords are found, the results from each of the recognizer plug-ins are received by the recognizer DLL. The results from the recognizer plug-ins are compiled into semantic categories by the recognizer DLL and the semantic categories are sent to the application program module. The application program module displays the semantic categories to the user in the electronic document.

These and other features, advantages, and aspects of the present invention may be more clearly understood and appreciated from a review of the following detailed description of the disclosed embodiments and by reference to the appended drawings and claims.

Brief Description of the Drawings

Fig. 1 is a block diagram of a computer that provides the exemplary operating environment for the present invention.

Fig. 2 is a block diagram illustrating an exemplary

architecture for use in conjunction with an embodiment of the present invention.

Fig. 3 is a flow chart illustrating a method for semantically labeling strings during creation of an electronic document in accordance with an exemplary embodiment of the present invention.

Fig. 4A is an illustration of a display of a preferred embodiment of a semantic category and its associated dropdown menu.

Figs. 4B and 4C are illustrations of a display of a preferred embodiment of a semantic category and its associated cascade menu.

Detailed Description

The present invention is directed toward a method and system for semantically labeling strings of text during creation of an electronic document and providing a selection of actions that may be performed based on the semantically labeled strings. A string is defined as a data structure composed of a sequence of characters usually representing human-readable text.

In one embodiment, the invention is incorporated into a suite of application programs referred to as "OFFICE", and more particularly is incorporated into a preferred word processing application program entitled "WORD 10.0" and a preferred spreadsheet application program entitled "EXCEL 10.0", both marketed by Microsoft Corporation of Redmond, Washington. Briefly described, the preferred application programs allow a user to create and edit electronic documents by entering characters, symbols, graphical objects, and commands.

In one embodiment, the invention comprises recognizing strings and annotating, or labeling, the strings with a type label. After the strings are annotated with a type label, application program modules may use the type label to provide users with a choice of actions. If the user's computer does not have any actions associated with that type label, the user may be

provided with the option to surf to a download Uniform Resource Locator (URL) and download action plug-ins for that type label.

Having briefly described an embodiment of the present invention, an exemplary operating environment for the present invention is described below.

Exemplary Operating Environment

Referring now to the drawings, in which like numerals represent like elements throughout the several figures, aspects of the present invention and the exemplary operating environment will be described.

Fig. 1 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. While the invention will be described in the general context of an application program that runs on an operating system in conjunction with a personal computer, those skilled in the art will recognize that the invention also may be implemented in combination with other program modules. Generally, program modules include routines, programs, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Fig. 1, an exemplary system for implementing the invention includes a conventional personal computer **20**, including a processing unit **21**, a system memory **22**, and a system bus **23** that couples the system memory to the

processing unit **21**. The system memory **22** includes read only memory (ROM) **24** and random access memory (RAM) **25**. A basic input/output system **26** (BIOS), containing the basic routines that help to transfer information between elements within the personal computer **20**, such as during start-up, is stored in ROM **24**. The personal computer **20** further includes a hard disk drive **27**, a magnetic disk drive **28**, e.g., to read from or write to a removable disk **29**, and an optical disk drive **30**, e.g., for reading a CD-ROM disk **31** or to read from or write to other optical media. The hard disk drive **27**, magnetic disk drive **28**, and optical disk drive **30** are connected to the system bus **23** by a hard disk drive interface **32**, a magnetic disk drive interface **33**, and an optical drive interface **34**, respectively. The drives and their associated computer-readable media provide nonvolatile storage for the personal computer **20**. Although the description of computer-readable media above refers to a hard disk, a removable magnetic disk and a CD-ROM disk, it should be appreciated by those skilled in the art that other types of media which are readable by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, and the like, may also be used in the exemplary operating environment.

A number of program modules may be stored in the drives and RAM **25**, including an operating system **35**, one or more application programs **36**, a word processor program module **37** (or a spreadsheet program module), program data **38**, and other program modules (not shown).

A user may enter commands and information into the personal computer **20** through a keyboard **40** and pointing device, such as a mouse **42.** Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit **21** through a serial port interface **46** that is coupled to the system bus, but may be connected by other interfaces, such as a game port or a universal serial bus (USB). A monitor **47** or other type of display device is also connected to the

system bus **23** via an interface, such as a video adapter **48**. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers or printers.

The personal computer **20** may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer **49**. The remote computer **49** may be a server, a router, a peer device or other common network node, and typically includes many or all of the elements described relative to the personal computer **20**, although only a memory storage device **50** has been illustrated in Figure 1. The logical connections depicted in Figure 1 include a local area network (LAN) **51** and a wide area network (WAN) **52**. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the personal computer **20** is connected to the LAN **51** through a network interface **53**. When used in a WAN networking environment, the personal computer **20** typically includes a modem **54** or other means for establishing communications over the WAN **52**, such as the Internet. The modem **54**, which may be internal or external, is connected to the system bus **23** via the serial port interface **46**. In a networked environment, program modules depicted relative to the personal computer **20**, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Fig. 2 is a block diagram illustrating an exemplary architecture **200** for use in conjunction with an embodiment of the present invention. The architecture includes an application program module **205**, such as word processor program module **37** (Fig. 1). The application program module **205** is able to communicate with a recognizer dynamic-link library **210** (hereinafter recognizer DLL) and an action dynamic-link library

215 (hereinafter action DLL) as a user is creating or editing an electronic document. The recognizer DLL **210** controls a number of recognizer plug-ins **220**. The action DLL **215** controls a number of action plug-ins **225**. The action DLL also controls a type-action database **230**.

In a preferred embodiment, the action plug-ins and recognizer plug-ins are Automation Servers. Automation Servers are well-known software components which are assembled into programs or add functionality to existing programs running on the Microsoft WINDOWS® operating system. Automation Servers may be written in a variety of computing languages and can be un-plugged from a program at run time without having to recompile the program.

The recognizer DLL **210** handles the distribution of strings from the electronic document running on the application program module **205** to the individual recognizer plug-ins **220**. The recognizer plug-ins **220** recognize particular strings in an electronic document, such as a word processing document or a spreadsheet document. The recognizer plug-ins **220** may be packaged with the application program module **205** or they may be written by third parties to recognize particular strings that are of interest. Typically, the recognizer DLL **210** passes strings to the recognizer plug-ins **220** in one paragraph or cell value increments.

As part of recognizing certain strings as including semantic information, the recognizer plug-ins **220** determine which strings are to be labeled and how they are to be labeled. After receiving these results from the various recognizer plug-ins **220**, the recognizer DLL **210** sends semantic categories to the application program module. In a preferred embodiment, a semantic category comprises the recognized string, a type label, and a download URL. A semantic category may also comprise metadata. The recognizer plug-ins **220** each run separately and the recognizer DLL **210** is responsible for handling the asynchronicity that results from different recognizer plug-ins

returning results with different delays.

After a string is labeled by a recognizer plug-in **220** and a semantic category is sent to the application program module **205**, the user of the application program module **205** will be able to execute actions that are associated with the type label of the semantic category. The action DLL **215** manages the action plug-ins **225** that are run to execute the actions. As with the recognizer plug-ins **220**, the action plug-ins **225** may be packaged with the application program module **205** or written by third parties to perform particular actions that are of interest to the third party. The action plug-ins provide possible actions to be presented to the user based upon the type label associated with the string. The action DLL **215** determines what type label the semantic category is and cross-references the type label in the type-action database **230** with a list of actions to determine what actions to present to the user. It should be understood that, in a preferred embodiment, the type-action database is not used. Instead, the list of actions is dynamically generated for each type by looking in the registry to determine which actions are installed and then querying the action DLLs to determine which types they apply to.

After the user chooses an action, the action DLL **215** manages the appropriate action plug-ins **225** and passes the necessary information between the action plug-ins and the application program module **205** so that the action plug-in may execute the desired action. Typically, the application program module sends the action DLL an automation request to invoke the action the user has selected.

As described above, the combination of the recognized string, type label, metadata and download URL is referred to herein as a semantic category. The type label is a semantic information label. The semantic category may also comprise metadata, which are hidden properties of the semantic category. An example of a semantic category may clarify the definition. Suppose a user enters the text "Gone With the Wind" into an electronic document. The string "Gone With the Wind"

may be identified as a semantic category of type label "Book Title" and of type label "Movie Title". In addition, metadata such as the ISBN number may be returned by the recognizer plug-in to the application program module as part of the semantic category. A download URL may be provided with the type labels "Book Title" and "Movie Title" in case the user's machine has not stored action plug-ins for these type labels. For example, an action for the type label "Book Title" may be "Buy this Book" from an online retailer. If the user does not have the action plug-in DLL **225** corresponding to "Buy this book", then the download URL may be used to navigate the user's web browser to an appropriate website to download this action plug-in.

It should also be understood that the present invention, in a preferred embodiment, also recognizes sequences of capitalized words that contain function words, and which are likely to be special, but for which there is no type label information. These strings are typically labeled by a grammar checker program module.

The actions provided for a semantic category may utilize both the type label and the text of the recognized string. For example, a word processor program module may use a grammar checker as a recognizer plug-in to label strings that are person names. After a string has been labeled as a person's name, the word processor program module may, through a standard user interface mechanism, allow users to execute pertinent actions, such as looking up the person's name in the contacts folder in a personal information manager program module, sending electronic mail, or searching for the person's name in an HR database.

Having described an exemplary architecture, an exemplary method **300** for semantically labeling strings during document creation will be described below in reference to Figs. 2 and 3.

Method for Semantically Labeling Strings During Document Creation

Fig. 3 is a flow chart illustrating a method **300** for

semantically labeling strings during creation of an electronic document in accordance with an exemplary embodiment of the present invention. Those skilled in the art will appreciate that this exemplary embodiment is a computer-implemented process that is carried out by the computer in response to input from the user and instructions provided by a program module.

Referring to Fig. 3, the method **300** begins at start step **305** and proceeds to step **310** when a user opens an electronic document in application program module **205**. In a preferred embodiment, the electronic document is a word processing document or a spreadsheet document. However, the invention is not limited to either of these specific types of electronic document.

At step **310**, the application program module **205** receives a new string, such as when the user enters a new paragraph into the electronic document or edits a previously entered paragraph. The method **300** then proceeds to step **315**.

At step **315**, the paragraph containing the new string is passed from the application program module **205** to the recognizer DLL **210**. The recognizer DLL is responsible for communicating with the application program module, managing the jobs that need to be performed by the recognizer plug-ins, receiving results from the recognizer plug-ins and sending semantic category information to the application program module. At boot time, the recognizer DLL communicates with its recognizer plug-ins to determine what languages it supports, what types it can apply, etc. It should be understood that, in a preferred embodiment, a paragraph is passed to the recognizer DLL at step **315**. However, in alternative embodiments, a sentence, the contents of a spreadsheet cell, a section of the document, the entire document, etc. may be passed to the recognizer DLL. In other words, the present invention is not limited to simply passing a paragraph to the recognizer DLL. The method **300** then proceeds to step **320**.

Still referring to step **315**, the application program

module **205** typically sends one paragraph at a time to the recognizer DLL. In addition, in a preferred embodiment, a grammar checker program module sends all semantic categories (without type labels) to the recognizer DLL that have been identified by the grammar checker program module. Passing these semantic categories (without type labels) to the recognizer DLL is important because doing so saves each recognizer plug-in from needing to decide whether something is a capitalized string interspersed with function words (a task that would require writing a number of regular expressions: Cap Cap Unc Cap; Cap Unc Cap; etc.). If a label is applied by a recognizer plug-in to a string the grammar checker program module labeled, the grammar checker label will then be removed.

At step **320**, during idle time, the paragraph (and information from the grammar checker program module) is passed to the recognizer plug-ins. The method then proceeds to step **325**.

It should be understood that, in a preferred embodiment, the recognizer DLL **210** maintains a job queue. If before the recognizer DLL **210** sends the paragraph to the recognizer plug-ins **220** the user edits the paragraph, then the job containing the edited paragraph is deleted and is not sent to the recognizer plug-ins. Then, a new job enters the queue at step **315** after the edited paragraph is received at step **310**. This job deletion is necessary to prevent the recognizer plug-ins from performing unnecessary work on a paragraph that has been edited.

At step **325**, the recognizer plug-ins are executed on the paragraph to recognize keywords or perform other actions defined by the recognizer plug-in. As part of executing the recognizer plug-in, the paragraph may be broken into sentences by the recognizer plug-in. However, each recognizer plug-in is responsible for its own sentence-breaking. After the keywords are found at step **325**, then the method proceeds to step **330**.

At step **330**, the results from each of the recognizer plug-ins are received by the recognizer DLL. The method then proceeds to decision step **335**.

At decision step **335**, it is determined whether the paragraph that has been reviewed by the recognizer plug-ins has been edited after the paragraph was sent to the recognizer DLL. If so, then the method **300** returns to step **315** and the edited paragraph is received by the recognizer DLL from the application program module. If not, then the method proceeds to step **340**.

At step **340**, the results from the recognizer plug-ins are compiled into semantic categories by the recognizer DLL and the semantic categories are sent to the application program module. At step **345**, the application program module displays the semantic categories to the user in the electronic document. The method **300** then ends at step **399**.

As understood from the above description, the present invention comprises an architecture for recognizing semantic categories that permits third parties to develop recognizer plug-ins to identify strings of one or more particular types. The recognizer plug-ins communicate with the application program module and receive a string from the application program module. The recognizer plug-ins may apply recognition algorithms to the string and communicate the identity of recognized strings back to the application program module.

After a string is labeled with a particular type label, the user will be able to execute action plug-ins that pertain to that type label. The action plug-ins preferably are COM objects that are executed via communication between the application program module and the action DLL. Parameters necessary to execute the action (the html of the string labeled as being of a particular type, the html of the string representing the current selection) will be passed from the application program module to the action DLL and, in turn, passed to the action plug-in.

Actions Assigned to Type Labels in a Preferred Embodiment

The present invention further comprises an architecture for identifying and executing a set of actions associated with a semantic category. This architecture comprises actions that apply to a particular type label (e.g. an action for book titles may be "Buy this book from shop.Microsoft.com") and executing those actions when the user so desires. An action is a user-initiated function applied to a typed string. For example, adding a name to the contacts folder is one action possible for a type label "Person name".

It should be understood that a significant aspect of the present invention is the power and flexibility that results from allowing third party vendors, such as IT professionals, to design and write recognizer plug-ins and action plug-ins for deployment within an organization or for deployment on the World Wide Web. Some example actions that may be executed include:

Schedule a meeting

Create task

Display calendar

Add to contacts folder

Look up in contacts folder, address book, Windows Address Book (WAB), Global Address List (GAL), etc.

Insert address into document

Send mail to

Display EXPEDIA map

Stock quote lookup

Send instant message to

In a preferred embodiment, different actions are assigned to different type labels and these type label-action assignments are stored in the type-action database **230**. Table 1 below illustrates some possible type label-action pairings.

Type Labels	Actions
Person name	Show contact info Add to contacts E-mail Insert address into document Send instant message to
Date	Show calendar for that day New task with that due date Schedule meeting that day
Place	Display EXPEDIA map Add to contacts
Address	Add to contacts
Phone number	Add to contacts
E-mail	Add to contacts
Date	Schedule a meeting
Task	Schedule a task
Meeting	Schedule a meeting

Table 1

For each type label, the type-action database **230** will store a download URL specified by the creator of the type label that users who do not have action-plug-ins or recognizer plug-ins for that semantic category type can go to in order to get action plug-ins and/or recognizer plug-ins. For example, the download URL for the type label "Book Title" might be microsoft.com/semanticcategories.asp. Once at that web page, a user may be offered downloads of various action plug-ins and recognizer plug-ins. There may also be an option on the user interface to navigate to the download URL so that recipients of documents with semantic categories can easily get the action plug-ins for those semantic categories.

Storing Semantic Categories

In a preferred embodiment, semantic categories are stored as part of the electronic document along with other

document information and are available when a document is transmitted from one computer to another computer. In a preferred embodiment, storing semantic categories in an electronic document is controlled by an "Embed semantic categories" checkbox. The checkbox is on by default. Turning it off will prevent semantic categories in the document from being saved. The state of the checkbox is per document. The same checkbox controls saving for both .htm and .doc documents.

Checking a "Save semantic categories as XML properties" checkbox (off by default) will write out the text of all of the semantic categories in the document and their labels in the header of the html file in XML (that is using the same tags as are used inline, but surrounded by <xml> And </xml>) for easy identification and parsing by search engines and knowledge management systems.

In a preferred embodiment, semantic categories are saved as a unique namespace plus a tag name. A namespace is an XML construct for uniquely identifying a group of XML tags that belong to a logical category. Thus, every semantic category is uniquely identified by its nametag (e.g., "streetname") in addition to its namespace (e.g., "schemas-microsoft-com:outlook:contact")

Although the method **300** described above is one method for identifying semantic categories, in a preferred embodiment of the present invention, there are two other mechanisms for identifying semantic categories. One mechanism is a grammar checker program module (not shown) connected to word processor program module **37**. Another mechanism is receiving a semantic category from another electronic document. For example, when text containing a semantic category is copied from one electronic document and pasted into another electronic document of the word processor program module **37**, the information identifying the semantic category is preserved and copied along with the copied text. These two additional mechanisms are described in more detail below.

Using a grammar checker program module to identify

semantic categories

In a preferred embodiment of the present invention, a grammar checker program module may be used to identify semantic categories of the following types:

- •Person names (Example: Bill Smith)
- •Complete dates (Example: May 1, 1999)
- •Partial dates (Examples: May 1999; May 1)
- •A limited subset of temporal expressions (Examples: today, tomorrow, next Tuesday)
- •Times (Examples: 4:00 PM; 17:30)
- •Addresses (Example: 1053 Church Street Abington, PA 19001)
- •Places (Example: Trafalger Square)
- •Cities (Example: Pittsburgh)
- •Phone Numbers (Example: 215-887-9093)
- •E-mail addresses (Example: bob@xyz.com)
- •Web addresses

In a preferred embodiment, the grammar checker program module will sometimes send a normalized form of the string to the word processor program module along with the other semantic category information. For example, for the strings May 11, 1999 and 05/11/1999, the grammar checker program module sends the normalized form "05/11/1999" to the word processor program module. The word processor program module stores this normalized form of the string to make it easier to query a personal information manager program module, web properties and other program modules. For example, the grammar checker program module may return normalized forms for the following strings:

- •Dates (Example: "today" normalized to "10/15/99")
- •Times (Examples: "4pm" normalized to "4:00 pm")
- •Telephone numbers (to normalize for variants involving parentheses and hyphens)

It should be understood that the normalized forms for dates and times will typically be determined at write time of a document rather than read time.

Identifying Semantic Categories via Cut and Paste

In a preferred embodiment, copying or cutting a semantic category from one application program module and pasting the semantic category into a second application program module that recognizes semantic categories will preserve the semantic category. In other words, copying and pasting (or dragging and dropping) a string labeled with semantic category information will copy the string and the string's semantic category information. Moreover, it follows that complete documents sent from one application program module (or computer) to another application program module (or computer) will also typically preserve the semantic categories.

In a preferred embodiment, semantic categories are placed in the clipboard in CF_HTML. CF_HTML, or clipboard format HTML, is essentially HTML with some clipboard specific properties such as the length of the HTML string. However, application program modules that do not recognize semantic categories may make a special effort to persist pasted semantic categories. For example, pasted semantic categories may be preserved as unknown HTML. semantic categories

Displaying Semantic categories to the User

Referring now to Fig. 4A, an illustration of a display of a preferred embodiment of a semantic category **400** and its associated dropdown menu **405** will be described. It should be understood that Fig. 4A is an illustration of a semantic category **400** and dropdown menu **405** as displayed to a user by the application program module **205**.

The string **410** associated with semantic category **400** is the string "Bob Smith". As shown in Fig. 4A, the string **410** of a semantic category **400** may be identified to the user by brackets **415**. Of course, many other devices such as coloring, underlining, icons, etc. may be used to indicate to the user that a particular string is a semantic category.

In a preferred embodiment, when the user hovers a cursor over the string **410** or places the insertion point within

string **410**, then dropdown menu **405** is displayed to the user. The dropdown menu typically displays a list of actions associated with a semantic category. The dropdown menu typically appears above and to the left of the semantic category string.

Typically, the first line of the dropdown menu indicates which string is the semantic category string (Bob Smith in Fig. 4A) and what type the semantic category is (Person name in Fig. 4A). Listed below the first line are actions **420** available for the semantic category type, such as "Send mail to...", "Insert Adress", and "Display contact information...".

The first item on the drop down menu below the seperator line is "Check for new actions..." **425**. "Check for new actions..." **425** will appear only for semantic categories whose download URL is aavailable to the application program module. If selected, "Check for new actions..." **425** uses the semantic category download URL to navigate the user's web browser to the homepage for the semantic category type applied to the string. For example, suppose new actions have been defined for the semantic category type "person name". If so, then new actions will be downloaded to the user's computer after selecting "Check for new actions..." **425**. "Check for new actions..." **425** will be grayed out if a download URL is unavailable for the semantic category.

If selected, the "Remove this semantic category" item **430** deletes the semantic category label from the string. If selected, the "Semantic categories" item **435** navigates the user to the semantic categories tab of the autocorrect dialog.

It should be understood that the application program module sends a request to the action DLL to determine which actions are shown with each semantic category type.

Actions Performed in Association with Semantic categories

There are a number of functions that users perform on typed data that preferred word processor program module **37** and semantic categories will make easier. The functions fall into two primary categories:

1) interacting with personal information manager contacts, tasks, meetings, and mail
2) interacting with properties on the World Wide Web or a corporate intranet

A single string may be associated with multiple semantic categories. Every semantic category has a type label with one or more action plug-ins defined for the type label. For example, the "Address" type label may have the "Open in Mappoint", "Find with Expedia Maps" and "Add to my Address Book" actions associated with it and each of these actions may have a different action plug-in to execute the action.

The actions assigned to a semantic category are assigned on a per type label basis, not on a per semantic category basis. For example, all semantic categories of type label "Address" will have the same actions assigned to them. The actions assigned to type labels also depends on the computer that the application program module is running on. Thus, if a computer has three actions registered for the type label "Address", then all strings with an "Address" type label will be assigned to three actions. However, if one of these semantic categories is sent to a computer which has only two actions registered for the "Address" type label, then the user will only be exposed to two actions for this semantic category.

Nesting of Semantic categories

In an embodiment of the present invention, semantic categories may be nested inside each other. For example, the string "George Washington" may include a semantic category with type label "Person Name" for the span "George Washington State" and a semantic category with type label "State" for the span "Washington". Moreover, two semantic categories may cover exactly the same span. For example, the string "George Washington" may include a semantic category with type label "Person Name" and a semantic category with type label "President".

Because the preferred application program module **37**

will support labeling a single string with multiple type labels (e.g. Bob Smith could be a semantic category labeled as a "Person Name" and labeled as a "Microsoft employee"), the preferred application program module **37** will use cascade menus on the dropdown menu if multiple semantic category types are assigned. Referring now to Figs. 4B and 4C, cascade menu **437** is illustrated.

The cascade menu **437** includes a list of the type labels included in string **412**. This list includes a type label "Person Name" **445** and a type label "Microsoft employee" **450**. The cascade menu further comprises the "Semantic categories..." item **435**.

As illustrated in Fig. 4B, by selecting the type label "Person Name" **445**, the user is presented with the actions associated with the type label "Person Name". These actions are similar to those described above with regard to Fig. 4A.

As illustrated in Fig. 4C, by selecting the type label "Microsoft employee" **450**, the user is presented with the actions associated with the type label "Microsoft employee". These actions are associated with the string "Bob Smith 42859" **455**, whereas the actions illustrated in Fig. 4B are associated with the string "Bob Smith". The only action shown in Fig. 4C is the action **460** to "Look up in HeadTrax...".

It should be understood that Figs. 4B-4C illustrate how a cascade menu may be used to allow the user to select which type label the user is interested in and to further select an action after selecting the type label.

In-document User Interface to Indicate Semantic categories

As described above with reference to Figs. 4A-4C, in a preferred embodiment, the application program module includes the option to display an in-document user interface to indicate the location of semantic categories. This in-document user interface may use a colored indication to indicate the location of a semantic category, such as the brackets **415** in Figs. 4A-4C. The in-document user interface will also be able to show nesting of

semantic categories. For example, if Michael Jordan is labeled as a semantic category with type label "Person Name", Michael is a semantic category with type label "First Name" and Jordan is a semantic category with type label "Last Name", the document may look like this with the brackets indicating semantic categories:

[[Michael][Jordan]]

Of course, the in-document user interface may be any sort of indication. For example, in the "EXCEL" spreadsheet application program, the interface comprises a triangle in the lower right hand portion of a cell to indicate that one or more semantic categories are present in the cell.

Invalidating the Semantic categories of a Paragraph of Text

Whenever the user edits a paragraph of text or a cell value in a document of the application program module, that text is flagged as dirty and the recognizer plug-ins and the grammar checker program module (if used) will need to be consulted again to identify semantic categories. The concept of dirtying text is well-known in the art.

In a preferred embodiment, only semantic categories recognized by the grammar checker program module are discarded when a paragraph of text is dirtied. That is, the application program module retains semantic categories identified by recognizer plug-ins as well as those pasted into the document even though the text has been dirtied.

Turning Off Annotation/Managing Recognizer Plug-ins

In a preferred embodiment, there is a check box in the application program module **37** that turns off all annotation of semantic categories. The radio switch is labeled "Label text with semantic categories as you type". It appears on the semantic categories tab of the autocorrect dialog. In this same dialog box, there will be a user interface to manage which recognizer plug-ins (including the grammar checker program module) are active.

The present invention allows tighter integration between names, dates, times, places, addresses, e-mail addresses and the information stored in e-mail and personal information

managers. For example, using the present invention, the user can type a contact name into their document, select "Insert address" from the user interface and the contact's address will be identified in the personal information manager and inserted directly into the document.

As another example of the flexibility of the invention's architecture, suppose a user wants to find a map to a client's business. With the architecture of the present invention, an address will be recognized as such and the user may select "Show map" from a user interface to automatically open the appropriate web page in the web browser and automatically communicate with the mapping service, such as by sending address information automatically to the mapping service without nay user input. The invention saves the user the step of entering the address information into the appropriate fields, thus saving the user time and eliminating potential transcription errors.

It should be understood that an important aspect of the present invention is that the identification of semantic categories is performed at write time rather than read time. That is, semantic categories are identified, or recognized, by recognizer plug-ins as the user enters information into the document. One reason this is important is that the recognition relies on the writer's computer rather than the reader's computer. This is important because the writer's computer may contain recognizer plug-ins that are not included in the reader's computer. The reader may be presented with the option of downloading an action plug-in or a recognizer plug-in when they receive the document from writer.

The present invention satisfies the need for a system capable of recognizing strings of text in documents on-the-fly and labeling the strings with semantic information. After this string is labeled, then a list of actions may be displayed to the user based upon the label associated with the string.

It should be understood that the recognizer DLL may distribute any amount of text to the recognizer plug-ins. The

invention has been described above in the context of the recognizer DLL sending a paragraph to a recognizer plug-in. However, any amount of text may be sent such as a sentence, a cell, a slide, etc.

Although the present invention has been described above as implemented in a word processing program module, it should be understood that the present invention may be implemented in other program modules, including, but not limited to, HTML authoring programs and programs such as the "POWERPOINT"® presentation graphics programand the "OFFICE" program module, both marketed by Microsoft Corporation of Redmond, Washington.

It should be understood that the recognizer DLL keeps track of pending work, such as the starting and ending position of the paragraph that is to be examined. During idle time, the recognizer DLL sends the paragraph to the recognizer plug-ins. If the user dirties a paragraph that has been sent to the recognizer plug-ins before the recognizer plug-ins return results, then the recognizer DLL marks the results as invalid. If the user dirties a paragraph after it has been checked for semantic categories, then the paragraph is re-checked by the recognizer plug-ins.

As described above, the semantic category may also include metadata returned by the recognizer plug-ins. For example, a recognizer plug-in that recognizes the titles of books may return as metadata an ISDN book number when it recognizes the title of a book. The ISDN book number metadata may then be used to provide actions. Metadata may also be used to disambiguate for actions and searches. For example, suppose a recognizer DLL is linked to a corporate employee database to recognize names. When the recognizer DLL recognizes "Bob Smith", it may store "employeeID=12345" as metadata in the background. Then, when an action is fired, the text in question will be known to reference Bob Smith, employee no. 12345 rather than Bob Smith, employee no. 45678. Also, the metadata may

allow searches to be performed independent of the actual text in a document. So, a search may be conducted on "Robert Smith" by looking for employee 12345 in the employee databases and by performing a search on the metadata for employee number 12345 to find documents with "Bob Smith" in them. There are also numerous other functions for metadata. For instance, DHTML could be inserted so special tricks may be performed within a web browser. Additionally, data used by other actions may be inserted such as someone's e-mail address that could be used by the send-mail-to action, a normalized version of the date could be stored to easily interact with a personal information manager, etc.

It should also be understood that limited semantic category information may be sent from the application program module to the recognizer plug-ins. For example, a grammar checker program module may be able to recognize a person's name to create a type label "Person Names". This type label information may be sent to recognizer plug-in for them to use. For example, a recognizer plug-ins that recognizes the names of famous people may not need to search every paragraph that is sent to it. The famous people recognizer plug-in may only need to search strings that have been recognized and labeled as "Person Names" and then determine whether any of these strings is the name of a famous person.

It should be understood that in a preferred embodiment the semantic categories are stored in the native file format, in XML, and in HTML. Of course, the semantic categories may be stored in any file format without departing from the spirit and scope of the present invention.

In one embodiment, the invention comprises annotating strings in documents with semantic information as the document is created. After a string is annotated with semantic information, a range of actions that utilize both the semantic information and the text of the string-so-labeled may be provided to the user. For example, a word processor program module may use a grammar checker program module to label person names.

After a string has been labeled as a person's name, the word processor program module may, through a standard user interface mechanism, allow users to execute actions pertinent to a person's name. For example, the actions may be searching for the person's name in a contacts folder, sending electronic mail to the person's e-mail address or looking up the person's name in a human resources database. Of course, the examples provided above are simply examples and should not be construed as limiting the present invention.

The architecture of the present invention is designed to allow IT departments or independent vendors to build recognizers for any of these things, which could be deployed within an organization to label these strings in documents. The IT department or service vendor could also create actions specific to these data types (such as checking the store room for a particular chemical or displaying a spec sheet for it; checking the order backlog for a particular car or the quantity on hand for a particular part; checking the schedule to see when a course is being taught or which department a faculty member is in.) It should be understood that recognizer plug-ins can be designed to perform arbitrary computation. For example, a recognizer plug-in could be designed to annotate words found on a list with a particular property (e.g. in order to recognize names of companies). Another recognizer plug-in may annotate stock ticker symbols (by looking for 3 to 5 letter words that are all capitalized). Another recognizer plug-in may query a database to see whether a particular string is a book title. The book title database may be on a local machine, on the corporate Intranet or halfway around the world and accessible via the Internet. The possibilities for different plug-in designs is endless. As a result of the possible differences between recognizer plug-ins, the recognizer plug-ins will run in a separate thread and the recognizer DLL will be responsible for handling the asynchronicity that will result from different recognizer plug-ins returning their results with different delays.

It should be understood that the present invention is designed to be able to function without any recognizer plug-in DLLs and any action plug-in DLLs. For example, if there are no recognizer plug-in DLLs, semantic categories may still be recognized by a grammar checker program module. If there are no action plug-in DLLs, then the user may be presented with a menu item that allows them to go to a download site to install an action plug-in DLL. The action plug-in DLL and recognizer plug-in DLL may also be combined together into one plug-in DLL.

In alternative embodiments, the recognizer plug-ins may be able to modify the content of a document upon recognition of a semantic category, such as bolding a company's name.

In alternative embodiments, the application program module may fire an event within its object model so that a plug-in that uses the object model could execute a piece of code when a semantic category is recognized (such as to display a special user interface when a semantic category is labeled).

In an alternative embodiment, the language of the text may be passed to the recognizer DLL or recognizer plug-ins so that analysis of the text may be skipped if the text is an unsupported language.

In alternative embodiments, word-breaking or sentence-breaking may be done outside of the recognizer plug-ins.

In alternative embodiments, the recognizer plug-ins may be able to examine the content of the document beyond the paragraph of text it was given. For example, the preceding line in a document may be used to determine whether something is really an address.

It should be understood that a recognizer plug-in is not limited to simply applying labels to text. Once a recognizer plug-in identifies a string as being of a particular type, it may enter the string into a database, send e-mail, start another application, etc.

Although the present invention has been described above as implemented in a preferred application program module, it will be understood that alternative embodiments will become apparent to those skilled in the art to which the present invention pertains without departing from its spirit and scope. Accordingly, the scope of the present invention is defined by the appended claims rather than the foregoing description.